Storage Access Characteristics of Computational Science Applications

Philip Carns,* Kevin Harms,† William Allcock,† Charles Bacon,† Samuel Lang,* Robert Latham,* and Robert Ross*
*Mathematics and Computer Science Division, †Argonne Leadership Computing Facility
Argonne National Laboratory
Argonne, IL 60439
*{carns,slang,robl,robr}@mcs.anl.gov, †{harms,allcock,bacon}@alcf.anl.gov

***Abstract*—Computational science applications are driving a demand for increasingly powerful storage systems. While many techniques are available for capturing the I/O behavior of individual application trial runs and specific components of the storage system, continuous characterization of a production system remains a daunting challenge for systems with hundreds of thousands of compute cores and multiple petabytes of storage. As a result, these storage systems are often designed without a clear understanding of the diverse computational science workloads they will support.**

In this study, we outline a holistic methodology for scalable, systemwide I/O characterization that combines storage device instrumentation and static file system analysis with a new mechanism for capturing detailed, application-level behavior. We demonstrate the effectiveness of our methodology by performing a multilevel, two-month study of Intrepid, a 557-teraflop IBM Blue Gene/P system. During that time, we captured application-level I/O characterizations from 6,481 unique jobs spanning 38 science and engineering projects with up to 163,840 processes per job. We also captured patterns of I/O activity in over 8 petabytes of block device traffic and summarized the contents of file systems containing over 191 million files.

From this collection of data we are able to quantify systemwide trends such as how application behavior changes with job size, the "burstiness" of the storage system, and the change in file system contents over time. We also identify the top ten storage users by application domain and investigate how their I/O strategies relate to I/O performance. One of these applications is then selected as a case study in I/O tuning based on integrated I/O characterization. We then use the results of our study to highlight trends that will affect the design of future storage systems, and we identify opportunities for improvement in I/O characterization methodology.

I. Introduction

Computational science applications are driving a demand for increasingly powerful storage systems. This situation is true especially on leadership-class systems, such as the 557 TFlop IBM Blue Gene/P at Argonne National Laboratory, where the storage system must meet the concurrent I/O requirements of hundreds of thousands of compute elements [1]. Hardware architecture, file systems, and middleware all play a key role in providing high-performance I/O capabilities in this environment. These components cannot be considered in isolation, however. The efficiency of the storage system is ultimately determined by the nature of the data stored on it and how applications choose to access that data. Understanding storage access characteristics of computational science applications is therefore a critical—and challenging—aspect of storage optimization.

A number of methods exist for analyzing application access characteristics and their effect on storage. Synthetic I/O benchmarks are easily instrumented and parameterized, but in many cases they fail to accurately reflect the behavior of scientific applications [2], [3], [4]. Application-based benchmarks are more likely to reflect actual production behavior, but the available benchmarks don't represent the variety of scientific domains and applications seen on leadership-class machines, each with unique access characteristics and data requirements. I/O tracing at the application, network, or storage level is another technique that has been successful in analysis of general-purpose network file systems [5], [6]. However, these tracing techniques are impractical for capturing the immense volume of I/O activity on leadership-class computer systems because of the overhead. Such systems utilize high-performance networks, generate workloads with concurrent I/O from thousands of processes, and are highly sensitive to any perturbation in performance.

As a result, there exist key gaps in our understanding of the storage access characteristics of computational science applications on leadership-class systems. To address this deficiency, we have developed an application-level I/O characterization tool, known as Darshan [7], that captures relevant I/O behavior at production scale with negligable overhead. We have deployed Darshan in conjunction with existing tools for block device monitoring and static file system anslysis in an effort to answer the following questions for a large-scale production system:

- What applications are running, what interfaces are they using, and who are the biggest I/O producers and consumers?
- How busy is the I/O system, how many files are being created of what size, and how "bursty" is I/O?
- What I/O interfaces and strategies are employed by the top I/O producers and consumers? How successful are they in attaining high I/O efficiency? Why?
- Can we use this data to actually help improve application I/O on Intrepid?

To answer these questions, we performed a long-running, multilevel I/O study of the Intrepid Blue Gene/P system at Argonne National Laboratory. The study spanned two months of production activity from January to March 2010. During that time we recorded three aspects of I/O behavior: storage device activity, file system contents, and application I/O characteristics. These three sources of information can be combined to present a comprehensive view of storage access characteristics and their relationships.

Using this data, we highlight the characteristics of ten of the most I/O-intensive projects running on the system. We also revisit one of these projects after completion of the study to show how integrated I/O characterization can be used to guide tuning and improve the performance of INCITE production applications.

The remainder of the paper is organized as follows. Section II describes the target system and the tools used to analyze its I/O activity. Sections III, IV, and V investigate the questions about I/O activity listed above. Section VI discusses whether we can use the data to tune an application based on our findings. Section VII summarizes related work, and Section VIII presents conclusions and avenues for future work.

II. Target system and methodology

This study was conducted on Intrepid, the IBM Blue Gene/P (BG/P) system at the Argonne Leadership Computing Facility (ALCF) at Argonne National Laboratory. The ALCF makes large allocations available to the computational science community via the Innovative and Novel Computational Impact on Theory and Experiment (INCITE) program [8]. Systems such as Intrepid therefore host a diverse set of applications from scientific domains including climate, physics, combustion, and Earth sciences.

Intrepid consists of a 163,840-core production system with 80 TiB of RAM and a peak performance of 557 TFlops. The primary high-performance storage system employs 128 file servers running both PVFS [1] and GPFS [9], with a separate, smaller home directory volume. Data is stored on 16 DataDirect Networks S2A9900 SANs. The storage system has a total capacity of 5.2 PiB and a peak I/O rate of approximately 78 GiB/s. The architecture and scalability of this storage system have been analyzed in detail in a previous study [1].

Intrepid groups compute nodes (CNs) into partitions of sizes 512, 1,024, 2,048, 8,192, 16,384, 32,768, and 40,960 nodes. All jobs must select one of these partition sizes, regardless of the number of nodes that will be used. Each set of 64 compute nodes utilizes a single, dedicated I/O forwarding node (ION). This ION provides a single 10 gigabit Ethernet link to storage that is shared by the CNs.

A. Characterizing storage device activity

On the storage side of the system, the DataDirect Networks SANs used by PVFS and GPFS are divided into sets of LUNs that are presented to each of the 128 file servers. Activity for both file systems was captured by observing traffic at the block device level. We recorded high-level characteristics such as bandwidth, amount of data read and written, percent utilization, and average response times.

Behavior was observed by using the *iostat* command line tool included with the Sysstat collection of utilities [10]. Iostat can report statistics for each block device on regular intervals. We developed a small set of wrappers (known as *iostat-mon*) to monitor iostat data on each file server. Data was collected every 60 seconds, logged in a compact format, and then post-processed to produce aggregate summaries. All local disk activity was filtered out to eliminate noise from operating system activity. This data was collected continuously over the entire period from Januray 23 to March 26, but four days of data were lost in February because of an administrative error.

B. Characterizing file system contents

The block device instrumentation described above captures all data movement, but it does not describe the nature of the persistent data that is retained on the system. To capture this information, we used the *fsstats* [11] tool. Fsstats analyzes entire directory hierarchies to collect a snapshot of static characteristics, such as file sizes, file ages, capacity, and a variety of namespace attributes. We ran the fsstats tool at the beginning and end of the study on both primary file systems. GPFS was measured on January 23 and March 25, while PVFS was measured on January 27 and March 31. This approach allowed us to observe the change in file system contents over the course of the study.

C. Characterizing application behavior

The most detailed level of characterization was performed by analyzing application-level access characteristics. Application-level characterization is critical because it captures I/O access patterns before they are altered by high level libraries or file systems. It also ensures that system behavior can be correlated with the specific job that triggered it. Application-level I/O characterization has traditionally been a challenge for arbitrary production workloads at scale, however. Tracing and logging each I/O operation become expensive (in terms of both overhead and storage space) at scale, while approaches that rely on statistical sampling may fail to capture critical behavior. We have developed a tool called *Darshan* [7] in order to bridge this gap. Darshan captures lossless information about each file opened by the application. Rather than trace all operation parameters, however, Darshan captures key characteristics that can be processed and stored in a compact format. Darshan instruments POSIX, MPI-IO, Parallel netCDF, and HDF5 functions in order to collect a variety of information. Examples include access patterns, access sizes, time spent performing I/O operations, operation counters, alignment, and datatype usage.

The data that Darshan collects is recorded in a bounded (approximately 2 MiB maximum) amount of memory on each MPI process. If this memory is exhausted, then Darshan falls back to recording coarser-grained information, but we have yet to observe this corner case in practice. Darshan performs no communication or I/O while the job is executing. It delays

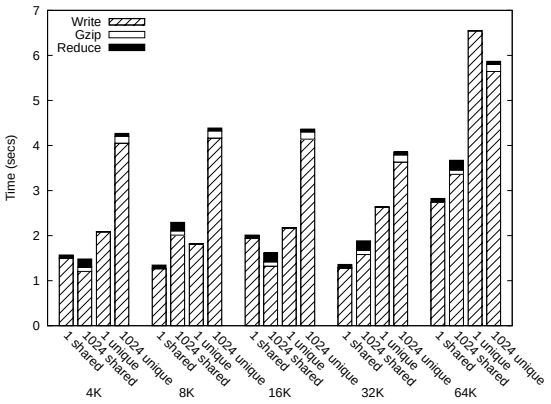

Fig. 1. Darshan output time for varying BG/P job sizes.

such activity until the job is shutting down. At that time Darshan performs three steps. First it identifies files that were shared across processes and reduces the data for those files into an aggregate record using scalable MPI collective operations. Each process then compresses the remaining data in parallel using Zlib. The compressed data is written in parallel to a single binary data file. Figure 1 shows the Darshan output time for various job sizes on Intrepid as measured in previous work [7]. This figure shows four cases for each job size: a single shared file, 1,024 shared files, one file per process, and 1,024 files per process. The largest case demonstrates that the Darshan shutdown process can be performed in less than 7 seconds even for jobs with 65,336 processes that opened 67 million files [7]. This time is not likely to be noticeable because jobs at this scale take several minutes to boot and shut down. In addition, we have measured the overhead per file system operation to be less than 0.05%, even for operations that read only a single byte of data.

Darshan was installed on Intrepid via modifications to the default MPI compilers. Users who built MPI applications using these default compilers were therefore automatically included in the study. Darshan did not achieve complete coverage of all applications, however. Some applications were compiled prior to the Darshan deployment on January 14, 2010. Other applications either did not use MPI at all or used custom build scripts that had not been modified to link in Darshan. Users do have the option of explicitly disabling Darshan at compile time or run time, though this option is rarely chosen.

To analyze the resulting data from Darshan, we postprocessed all log files and loaded the resulting data into a unified SQL database. We also utilized a graphical summary tool included with Darshan to generate summary reports for particular jobs of interest. This same tool is available to users and system administrators; it enables immediate feedback on the I/O behavior of any production job, in many cases eliminating the need to explicitly instrument or rerun jobs in order to troubleshoot I/O performance problems.

D. Performance metrics

One important metric for applications is *aggregate I/O bandwidth*. For parallel I/O benchmarks we typically calculate this by dividing the amount of data moved by the time of the slowest MPI process, with some coordination ensuring that I/O overlapped. In this work we are observing real applications running in production, and these applications may not have the same coordination seen in I/O benchmarks. Additionally, because the applications are running across a range of job sizes, it is useful for comparison purposes to examine performance relative to job size, rather than as an absolute value. We introduce a generic metric for read and write performance that can be derived from the Darshan statistics of unmodified applications and scaled across a variety of job sizes.

TABLE I
PERFORMANCE METRICS FOR IOR EXAMPLES

Example	Actual	Estimated	
	MiB/s	MiB/s	MiB/s/CN
IOR N-1 write	4021.01	4026.91	3.93
IOR N-1 read	6050.73	6067.70	5.93
IOR N-N write	3957.88	4050.39	3.96
IOR N-N read	5877.41	5883.73	5.75

Darshan records independent statistics for each file accessed by the application, including the number of bytes moved, cumulative time spent in I/O operations such as `read()` and `write()`, and cumulative time spent in metadata operations such as `open()` and `stat()`. The aggregate I/O bandwidth can be estimated by dividing the total amount of data transferred by the amount of I/O time consumed in the slowest MPI process. To make comparisons across jobs of different sizes, we divide the aggregate performance by the number of compute nodes allocated to the job. The result is a MiB per second per compute node (MiB/s/CN) metric, calculated as follows:

$$MiB/s/CN = \left(\frac{\sum_{rank=0}^{n-1}(bytes_r + bytes_w)}{max_{rank=0}^{n-1}(t_{md} + t_r + t_w)} \right) / N_{cn}.$$

In this equation, n represents the number of MPI processes, while N_{cn} represents the number of compute nodes. Intrepid has four cores per compute node, so those two numbers seldom match. Here $bytes_r$ and $bytes_w$ represent the number of bytes read and written by the MPI process, respectively, while t_{md}, t_r, and t_w represent time spent in metadata, read, and write operations, respectively. A slight variation is used to account for shared files because, in that scenario, Darshan combines statistics from all MPI processes into a single record and, in doing so, loses track of which process was the slowest.[1] For shared files we therefore estimate the I/O time as the elapsed time between the beginning of the first `open()` call and the end of the last I/O operation on the file.

To verify the accuracy of this approach, we used the IOR benchmark. The results are shown in Table I. All IOR jobs used 4,096 processes and transferred a total of 1 TiB of data to or from GPFS. Our examples used both shared files and unique files per process (notated as N-1 and N-N, respectively). The aggregate performance derived from Darshan deviated by less than 3% from the value reported by IOR in each case. Note

[1]This situation has since been addressed in Darshan 2.0.0.

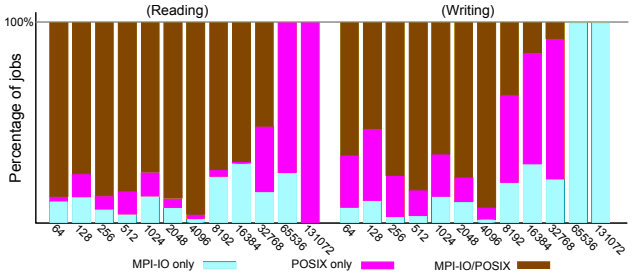

Fig. 2. Interfaces used by jobs as a function of job size.

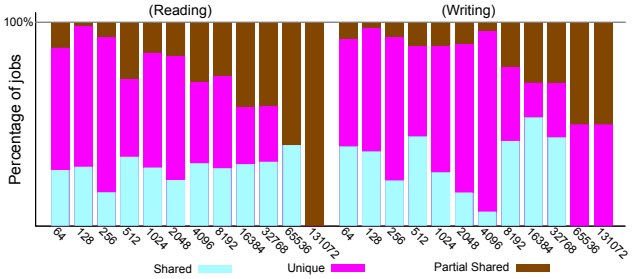

Fig. 3. I/O strategy used by jobs as a function of job size.

that reads obtained higher performance than writes, likely due to current SAN configuration settings that disable write-back caching but still cache read operations.

IOR, as configured in these examples, issues perfectly aligned 4 MiB operations concurrently on all processes. Table I therefore also indicates the approximate maximum MiB/s/CN that can be observed on Intrepid. The general maximum performance is bound by the network throughput the ION can obtain. The BG/P tree network, which connects CN and ION, supports approximately 700 MiB/s, which gives a theoretical maximum of 10.94 MiB/s/CN. The ION to storage network supports approximately 350 MiB/s, which results in a theoretical maximum of 5.47 MiB/s. With GPFS, a read workload can take advantage of read-ahead and caching to get above the storage network maximum. One final caveat is that the maximum possible MiB/s/CN rate will diminish as the total performance approaches the limit of the file system [1]. The theoretical maximum performance for a 40,960-node, 163,840-process job is 1.59 MiB/s/CN. These metrics are not perfect representations of I/O performance; however, with the exception of three jobs that achieved unreasonably high measures by our estimate, the metric provides meaningful insight into relative performance for production jobs that cannot otherwise be explicitly instrumented. Further analysis indicates that the three outliers exhibited very sparse, uncoordinated I/O that did not fit the model.

III. Application Trends and I/O-Intensive Projects

In the time period from January 23 to March 26, Intrepid executed 23,653 jobs that consumed a total of 175 million core-hours. These jobs were divided into 66 science and engineering projects (not counting maintenance and administration). Of these, 37 were INCITE projects, as described in Section II. The remainder were discretionary projects that are preparing INCITE applications or porting codes to the Blue Gene/P architecture. Of that total workload, Darshan instrumented 6,480 (27%) of all jobs and 42 million (24%) of all core-hours. At least one example from 39 of the 66 projects was captured.

A. Overall application trends

As part of our analysis we investigated the overall trends of the applications instrumented through Darshan on Intrepid. We were interested primarily in two characteristics. First, we wanted to discover which I/O interfaces were used by applications at various job sizes. High-level interfaces such as PnetCDF and HDF5 ease the data management burden and provide data portability, but it has remained unclear how many applications utilize these interfaces and how much data is moved through them. MPI-IO provides useful optimizations for accessing the parallel file systems deployed at leadership-class supercomputing centers, but applications may continue to use POSIX interfaces for a variety of reasons.

Second, we wanted a clearer understanding of the patterns of access in these applications. We focused on the type of file access at various job sizes, looking at the frequency and amount of I/O done to unique files (also known as N:N), shared files (N:1), and partially shared files (only a subset of processes perform I/O to the file, N:M where M $<$ N). Intuitively, applications performing I/O across a large number of processes will be more likely to use shared files in order to ease the file management burden that results when a large job writes 10,000–100,000 files, but it has remained unclear when applications choose shared files. Also, various studies have shown that at the largest job sizes, some file systems perform better under unique file workloads because lock contention is avoided [12], while others perform better under shared file workloads because they ease the metadata management burden [1]. We wanted to see how application performance varied according to the strategy chosen.

Figures 2 and 3 give an overview of the I/O interfaces and access patterns used by applications at various job sizes. We see in Figure 2 that the POSIX interfaces were used by the majority of jobs and performed the bulk of the I/O, especially for reading and at smaller process counts. Some applications used MPI-IO, particularly at the highest process counts and for applications that primarily wrote data. So few of the applications in our study used high-level libraries that they would not have been visible in the graph.

Figure 3 shows that the I/O strategy used by jobs varied considerably depending on the size of the job. Unique files were the most common access method for small jobs, while partially shared files were most common access method for large jobs. In terms of quantity of data, most I/O was performed to files that were shared or paritially shared. The widespread use of partially shared files indicates that applications are not relying on MPI-IO collective buffering optimization but, rather, are performing their own aggregation by writing and reading shared files from subsets of processes.

B. I/O-intensive projects

Not all of the 39 projects captured by Darshan were significant producers or consumers of data. Figure 4 illustrates how much data was read and written by the ten projects that moved the most data via Darshan-enabled jobs. The projects are labeled according to their general application domain. The first observation from this figure is that that a few projects moved orders of magnitude more data than most others. The project with the highest I/O usage, EarthScience, accessed a total of 3.5 PiB of data. Another notable trend in Figure 4 is that eight of the top ten projects read more data than was written. This is contrary to findings of previous scientific I/O workload studies [13]. By categorizing the data by project we see that the read/write mix varies considerably by application domain.

Table II lists coverage statistics and application programmer interfaces (APIs) used by each of the projects shown in Figure 4. Darshan instrumented over half of the core-hours consumed by seven of the ten projects. NuclearPhysics, Chemistry, and Turbulence3 were the exceptions and may have generated significantly more I/O activity than is indicated by Figure 4. The fourth column of Table II shows which APIs were used directly by applications within each project. P represents the POSIX `open()` interface, S represents the POSIX stream `fopen()` interface, M represents MPI-IO, and H represents HDF5. Every project used at least one of the two POSIX interfaces, while four projects also used MPI-IO. Energy1 notably utilized all four of HDF5, MPI-IO, POSIX, and POSIX stream interfaces in its job workload.

This subset of projects also varies in how many files are used. Figure 5 plots the number of files accessed by application run according to its processor count for our ten most I/O-intensive projects. If several application instances were launched within a single job (as is common on Intrepid), each instance is shown independently. Reinforcing Figure 3, we see four rough categories: applications that show an N:N trend, ones that show an N:1 trend, a group in the middle exemplified by Turbulence3 that are subsetting (N:M), and a fourth category of applications operating on no files! The large number of application runs that operated on zero files is surprising. Darshan does not track standard output or standard error. One possible explanation is that projects appear to run

TABLE II
DARSHAN COVERAGE OF HIGHLIGHTED PROJECTS

Project	Job Coverage	Core-Hour Coverage	APIs
EarthScience	779/1488	10.9/11.8 M	S,P
NuclearPhysics	1653/6159	11.3/62.7 M	P
Energy1	994/1340	3.7/5.7 M	H,M,S,P
Climate	32/130	2.0/3.3 M	S
Energy2	384/1433	3.9/4.4 M	S,P
Turbulence1	242/467	2.6/4.6 M	M,S,P
CombustionPhysics	15/42	1.8/2.4 M	S,P
Chemistry	28/144	0.1/0.6 M	S
Turbulence2	70/157	0.3/0.3 M	M,P
Turbulence3	172/418	0.1/13.3 M	M,S,P

API: P = POSIX, S = POSIX stream, M = MPI-IO, H = HDF5

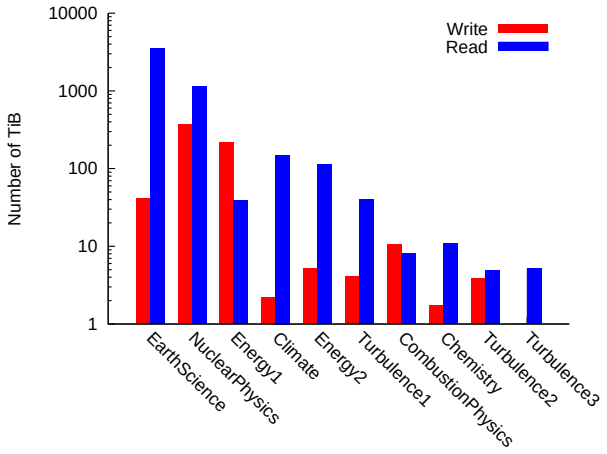

Fig. 4. Data moved per project in Darshan-enabled jobs.

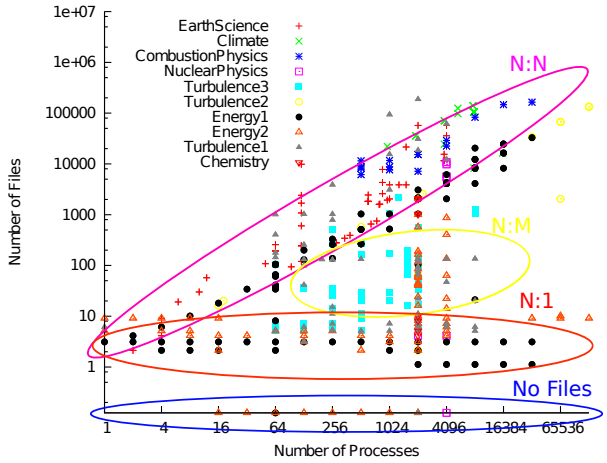

Fig. 5. Number of files written by size of application run for our ten most I/O-intensive projects.

a few debug jobs to run diagnostic or preliminary tests that write results only to standard out or standard error and then proceed to run "real" jobs.

Of the N:N applications, some access as many as 100 files per process. Programs accessing multiple files per process might need special attention when scaling to full-machine runs because of challenges in metadata overhead and file management.

Figure 5 also demonstrates that some projects have both N:1 and N:N jobs. Perhaps the clearest example is NuclearPhysics, the purple rectangle, about which more will be said in Section V-B.

IV. STORAGE UTILIZATION

The previous section provided an overview of how applications and jobs of varying sizes interacted with the storage system. In this section we investigate how this interaction translates into utilization at the storage device and file system level.

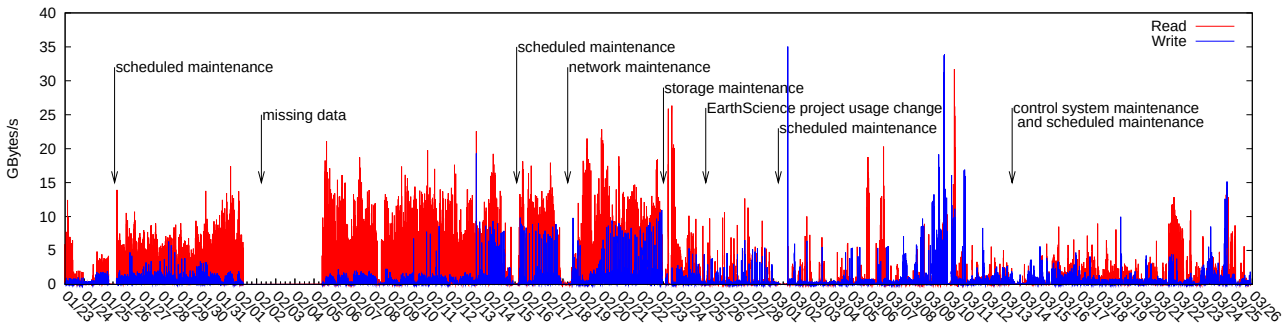

Fig. 6. Aggregate throughput on one-minute intervals.

Figure 6 shows the combined aggregate throughput at the block device level of Intrepid's main storage devices from January 23 to March 26. This includes both GPFS and PVFS activity. It also includes interactive access from login nodes as well as analysis access from the Eureka visualization cluster. Notable lapses in storage activity have been correlated with various maintenance windows and labeled accordingly. There were four notable scheduled maintenance days, as well as three unplanned maintenance windows due to network, storage, or control system issues. Note that the data ranging from 9:00 am February 1 to 10:00 am February 5 was lost because of administrative error, but the system was operating normally during that time.

The peak read throughput achieved over any one minute interval was 31.7 GiB/s, while the peak write throughput was 35.0 GiB/s. In previous work, we found that end-to-end throughput on this system varied depending on the access pattern [1]. In that study we measured maximum read performance from 33 to 47 GiB/s and maximum write performance from 30 to 40 GiB/s, both using PVFS. The system did not quite reach these numbers in practice during the interval shown in Figure 6. The reason may be SAN hardware configuration changes since the previous study. Our previous study also took advantage of full system reservations during Intrepid's acceptance period, with no resource contention.

From the iostat logs we can also calculate the amount of data moved over various time intervals. An average of 117.1 TiB were read per day, and 31.5 TiB were written per day. A total of 6.8 PiB and 1.8 PiB were read and written over the study interval, not counting the four missing days of data. Although reads made up 78.8% of all activity over the course of the study, this was largely due to the behavior of a single project. The EarthScience project was noted in Section III for having the most read-intensive workload of all projects captured by Darshan. It read over 3.4 PiB of data during the study, or approximately half of the total read activity on the system. We investigated that project's usage activity in scheduler logs and found that it significantly tapered off around February 25. This corresponds to a visible change in the read/write mixture at the same time in Figure 6. For the following two weeks, reads accounted for only 50.4% of all I/O activity.

We also note that some fraction of read activity was triggered by unaligned write accesses at the application level. Both

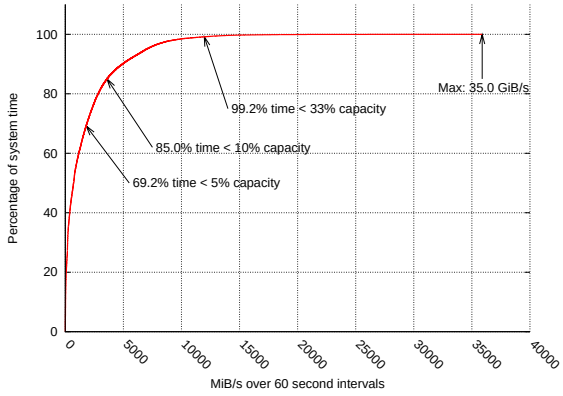

Fig. 7. Cumulative distribution of aggregate storage throughput.

GPFS and PVFS must ultimately perform read/modify/write operations at the block level in order to modify byte ranges that do not fall precisely on block boundaries. As we will see in Section V, unaligned access is common for many applications.

Figure 6 suggests that the I/O activity is also bursty. To quantify this "burstiness," we generated a cumulative distribution function (CDF) of the combined read and write throughput on the system for all 63,211 one-minute intervals recorded from iostat. The result is shown in Figure 7. The average total throughput was 1,984 MiB/s. The peak total throughput was 35,890 MiB/s. For 98% of the time, the I/O system was utilized at less than 33% of peak I/O bandwidth. This matches with the common understanding of "burstiness" of I/O at these scales. Because leadership-class I/O systems are provisioned to provide a very high peak bandwidth for checkpointing, it follows that during computation phases the I/O system will be mostly idle. An untapped opportunity exists for storage and I/O architectures that can take advantage of these idle periods.

A. File system contents

Despite the quantity of data that was transferred through the storage system on Intrepid, a surprising amount of it was stored in relatively small files at the file system level. Figure 8 illustrates the cumulative distribution function of file sizes in March 2010. The most popular size range was 64 KiB to 128 KiB, with over 71 million files. Of all files, 86% were under 1 MiB, 95% were under 2 MB, and 99.8% were under 128 MB. The largest file was exactly 16 TiB.

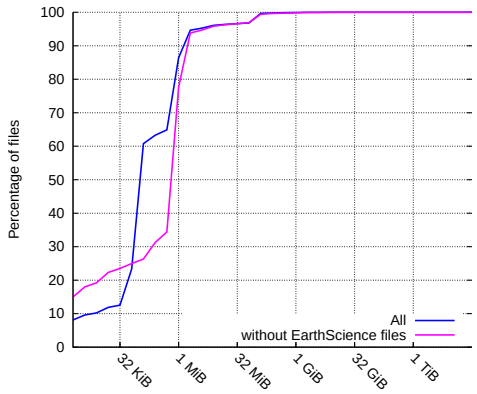

Fig. 8. Cumulative distribution of file sizes, March 2010.

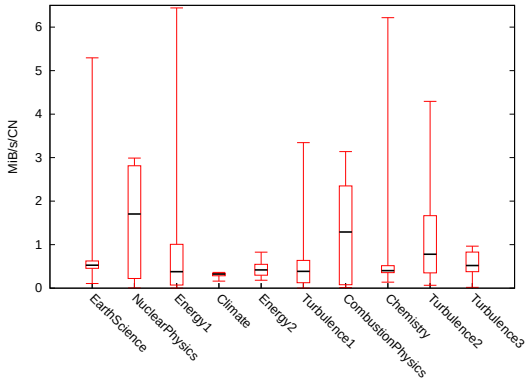

Fig. 10. Performance per project in large Darshan-enabled jobs (EarthScience outliers are not shown). Peak performance falls between 1.6 MiB/s/CN and 10.9 MiB/s/CN, depending on job size and other active jobs.

Closer investigation, however, revealed that a single project, EarthScience, was significantly altering the file size characteristics. The second line in Figure 8 shows what the distribution of file sizes would have been without counting the files belonging to that project. Without EarthScience's files, the most popular file size would have been 512 KiB to 1 MiB and only 77% of the files would have been under 1 MiB in size.

We can also observe how the file systems changed over the study interval by comparing fsstats results from the beginning and end of the two-month study. Table III shows growth of the primary file systems on Intrepid, in terms of both total capacity and number of files. Over the study period, the number of files doubled. Just as in the static analysis, however, this growth was largely the result of the EarthScience project data. EarthScience was responsible for 88% of the additional files created during the study period but generated only 15% of the new data by capacity. If the number of files continues to increase at the same rate, the file system will reach 1 billion files in September, 2011.

B. Overwriting data

Over 80% of the data stored on Intrepid has not been modified in over 64 days, and over 70% of it has not been modified in over 128 days. These results suggest that data is rarely overwritten once it is stored on the file system. Darshan characterization supports this observation as well. We found that of the 209.5 million files written by jobs instrumented with Darshan, 99.3% either were created by the job or were empty before the job started. This data suggests that replication, compression, and hierarchical data management algorithms that take advantage of the presence of infrequently modified files are applicable in this environment.

We note that the jobs characterized by Darshan wrote more files in the two-month study than were actually present in the file system at the end of the study. This fact, in conjunction with the earlier observations, indicates that files are either deleted within a relatively short time frame or else stored unchanged for extended periods of time.

TABLE III
FILE SYSTEM STATISTICS

	Total		EarthScience	
Date	Usage	Files	Usage	Files
January	1.14 PB	96.2 M	274.4 TB	12.3 M
March	1.28 PB	191.4 M	295.1 TB	96.2 M
Change	+12.3%	+99.0%	+7.5%	+682.1%

V. I/O CHARACTERISTICS BY PROJECT

We have established that the I/O workload on Intrepid consists of a variety of access patterns and file usage strategies and that the underlying storage system experiences bursts of I/O demand. In this section we explore in greater detail how storage access characteristics vary by application domain and how those characteristics correlate with I/O performance.

Figure 10 is a box plot of performance measured by using the MiB/s/CN metric outlined in Section II-D. We have filtered the jobs captured by Darshan to include only those that used at least 1,024 processes and moved at least 500 MiB of data. This approach eliminates noise from jobs that moved trivial amounts of data. All statistics shown in the remainder of this study are filtered by the same criteria. For each project in Figure 10, we have shown the minimum, median, and maximum, as well as the Q1 and Q3 quartiles. Some projects exhibited very consistent performance, while others varied over a relatively wide range. Very few jobs from any project approached the maximum values established in Section II-D.

Table IV summarizes a set of key storage access characteristics as averaged across jobs within that project. The MiB/s/CN and metadata overhead are computed as described in Section II-D. The second column shows percentage of cumulative I/O function time (across all processes) that was spent performing metadata operations rather than `read()` or `write()` operations. This value is much higher than expected in some cases because of the GPFS file system flushing writes to small files at `close()` time, because Darshan counts all `close()` operations as metadata. The other columns show the number of files accessed and created per MPI process, the percentage of sequential and aligned accesses, and the amount of data moved per process. Often accesses by a given process are highly sequential, as has been seen in previous studies [13]. Figure 9 illustrates the access sizes used by each project

TABLE IV
AVERAGE I/O CHARACTERISTICS OF LARGE JOBS BY PROJECT

Project	MiB/s/CN	Cumulative md cost	Files per proc	Creates per proc	seq.	aligned	MiB per proc
EarthScience	0.69	95%	140.67	98.87	64%	97%	1779.48
NuclearPhysics	1.53	55%	1.72	0.63	100%	0%	234.57
Energy1	0.77	31%	0.26	0.16	87%	36%	66.35
Climate	0.31	82%	3.17	2.44	97%	5%	1034.92
Energy2	0.44	3%	0.02	0.01	86%	11%	24.49
Turbulence1	0.54	64%	0.26	0.13	77%	25%	117.92
CombustionPhysics	1.34	67%	6.74	2.73	100%	0%	657.37
Chemistry	0.86	21%	0.20	0.18	42%	47%	321.36
Turbulence2	1.16	81%	0.53	0.03	67%	50%	37.36
Turbulence3	0.58	1%	0.03	0.01	100%	1%	40.40

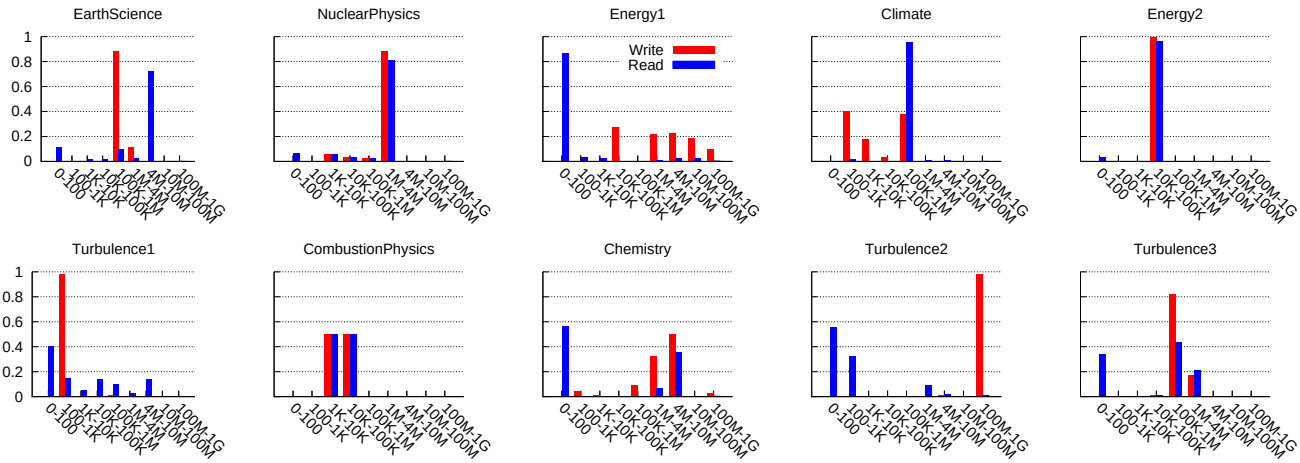

Fig. 9. Access sizes of large jobs by project.

in greater detail, using histograms to represent the percentages of accesses that fell within each range.

The remainder of this section will refer back to Table IV and Figure 9 as we explore the characteristics of each project in greater depth.

A. EarthScience

The EarthScience project has already featured prominently in previous sections, because it dominated both the read activity and number of files stored on Intrepid during the study interval. Despite the high level of I/O usage, however, EarthScience ranked near the low range of median I/O performance. Other than the three outliers discussed earlier, the performance is consistent, with an interquartile range (IQR) of only 0.17 MiB/s/CN. Further inspection indicated that the EarthScience workload is dominated by 450 nearly identical jobs, each of which utilized 4,096 processes. These jobs were often further subdivided into a sequence of up to 22 repeated instances of the same application within a job allocation. Each instance accessed approximately 57,000 files, leading some jobs to access a total of more than *1 million* distinct files over the lifetime of the job.

The EarthScience project read over 86 times as much data as it wrote. The data that it did write, however, was broken into a large number of newly created files. Of the 141 files accessed per process on average, 99 were created and written by the job itself. As noted in Section IV, this project alone contributed over 96 million files to the 191.4 million stored on Intrepid at the end of the study. The direct result of splitting data into so many files is that each job spent more of its I/O time performing metadata operations than actually reading or writing application data. Over 20 TiB of data were written into files averaging 109 KiB each in size, leaving the file system little opportunity to amortize metadata overhead. The apparent metadata cost is exaggerated somewhat by I/O time that is attributed to `close()` rather than `write()`, but that doesn't change the fact this metadata overhead is a limiting factor in overall I/O efficiency for the project.

B. NuclearPhysics

NuclearPhysics exhibited the widest IQR of job performance of any of the ten most I/O-intensive projects. This variability was not caused by fluctuations in performance of a single application. Two applications with different I/O characteristics were run by users as part of this project. In one set, 809 nearly identical jobs accounted for the upper quartile and were among the most efficient of any frequently executed application during the study. In the other set, 811 jobs accounted for the lower quartile. This example illustrates that access characteristics may vary significantly even across applications from the same domain on the same system.

The faster of the two applications utilized a partially shared file access pattern (N:M) and was atypical among jobs observed in this study because many of its files were both read and written to during the same job. The metadata overhead of creating and writing multiple files was amortized

by the quantity of I/O performed to each file. An example job read 1.38 TiB of data and wrote 449.38 GiB of data. This job is also a clear example of a behavior that was first speculated in Section III-A, namely, that some applications are implementing their own form of I/O aggregation rather than using the collective functionality provided by MPI-IO. This particular application used POSIX exclusively and was run with 4096 processes, but the first 512 MPI ranks performed all of the I/O for each job.

The slower of the two applications that dominated this project presents an example of an application that performs "rank 0" I/O, in which a single process is responsible for all of the I/O for the job. In this case the jobs were either 2,048 or 4,096 processes in size. The fact that all I/O was performed by a single rank resulted in a MiB/s/CN score as low as 0.2 in most cases. At first glance this appears to be very poor I/O behavior, but in practice these jobs read only 4 GiB of data, and the time to read that data with one process often constituted only 1% of the run time for this application. So while the storage access characteristics were poor, it will likely not be a significant problem unless the application is scaled to a larger problem size. This application accounted for the earlier observation in Figure 5 that NuclearPhysics exhibited both N:M and N:1 styles of access patterns.

C. Energy1

The performance fluctuation in the Energy1 project results from variations within a single application that used different file systems, job sizes, APIs, data sizes, and file sharing strategies. Discussions with the scientists involved in the project revealed that this behavior was the result of experimental I/O benchmarking and does not represent production application behavior. However, it was interesting to capture an application-oriented I/O tuning experiment in progress.

D. Climate

The Climate project executed 30 jobs. The jobs tended to use co-processor mode, which means 2 MPI processes per node with 2 threads per MPI process. The application performance was likely dominated by three factors. One, each process created two files, translating to a higher metadata overhead. Two, the application performed a seek for every read/write operation. All seeks need to be forwarded to the ION to be processed, making the calls unusually expensive relative to a cluster system; 82% of I/O time was spent in metadata. Three, write operations were only 64 KiB when the file system block size is 4 MiB. Writes this small are not efficient on this system.

E. Energy2

The Energy2 project executed 58 jobs at one of two sizes, 2,048 or 4,096 processes. The overall time spent in I/O as a whole was very small (less than 1%). The I/O performance of this project was low compared to the others even though it had low overhead for metadata. The performance loss was due to small independent writes (less than 10 KiB) that occurred only on rank 0. This project does utilize a single, shared file for reading, in which all processes read significantly more bytes than are written, and at a larger access size, producing very good performance. Given the small amount of time spent in I/O as compared to the overall application run-time and the minimal number of bytes, maximizing the write performance doesn't seem to be a priority.

TABLE V
TURBULENCE1 PERFORMANCE BY JOB

Job	Procs	Performance (MiB/s/CN)	Metadata (%)	Key Access Size
A	8192	2.92	28%	74KiB
B	2048	1.12	2%	20B (w)
C	1024	0.64	6%	$<$ 20B (w)
D	4096	0.38	41%	$<$ 20B (w)
E	2048	0.11	36%	$<$ 700B (r+w)
F	32768	0.0009	1%	4B (r)

F. Turbulence1

The Turbulence project sample contained 118 diverse jobs. The jobs were run by three users, with process counts between 1,024 and 32,768. Each user ran a few different applications, which led to a wide performance range for all applications. Table V details example jobs at the different performance scales.

All of these jobs had a common I/O profile. Each application used shared files as well as unique files on a file-per-process basis. For applications that needed to read/write more substantial amounts of a data, a single shared file was used with a 4 MiB access size. The bulk of the accesses, however, were very small read or write operations. As a result, performance was determined by the effectiveness of either collective I/O or POSIX stream operations to combine these small I/O operations into larger requests. The fastest job performed the bulk of its I/O to shared files using POSIX read operations using the fast scratch file system. The access size was not particularly large but probably benefited from the GPFS read-ahead caching. The slowest application used a large number of independent POSIX reads of a very small access size, on the order of four bytes to the slower home file system.

G. CombustionPhysics

The CombustionPhysics project comprised only 11 jobs in the sample based on the selection criteria. Within those 11 were a wide variety of different-sized jobs. The size of the job had a significant impact on the I/O rate.

This project appeared to be studying strong-scaling since the total number of bytes transferred for each job was similar regardless of size. Hence, the bytes per process transferred were smaller at each larger job size. At the two smaller node counts (1,024 and 2,048) the total I/O time for the job was small ($<$ 1%) compared to the total compute time. At the 4,096 node count, however, the total I/O time became 40% of the run time, and the percentage of time spent in metadata exploded to 35%. At each larger node count the percentage of

TABLE VI
COMBUSTIONPHYSICS PERFORMANCE BY JOB SIZE

Nodes	Jobs	Performance MiB/s/CN
1024	3	3.0
2048	3	1.3
4096	2	0.36
8192	1	0.08
16384	1	0.03
32768	1	0.01

I/O time spent in metadata increases, eventually topping out at 99%.

Table IV indicates that this project created about three files per process on average. This I/O strategy did not scale well on Intrepid for higher processor counts and smaller amounts of data per process (see Table VI). In Section VI we will revisit this project and evaluate the impact of various I/O tuning strategies that were guided by the findings of our integrated I/O characterization methods.

H. Chemistry

All of the data captured for the Chemistry project corresponds to a single application. The bulk data accesses from the application were all perfectly aligned to the file system at 4 MiB. The metadata overhead was also low, because the majority of jobs accessed fewer than 10 total files regardless of job size. The I/O efficiency is poor despite these characteristics, however. The reason is that (with one exception) all of the Chemistry jobs captured by Darshan performed I/O exclusively from a single process, regardless of the size of the job. These jobs achieved performance similar to the lower quartile jobs of the NuclearPhysics project that utilized the same strategy.

One instance of the same application was executed with notably different characteristics. The job size in that case was 2,048, and half of the processes were involved in performing I/O. As in the upper quartile NuclearPhysics cases, the application appears to be manually performing aggregation on behalf of the other processes, as no MPI-IO is involved. The 1,024 I/O tasks combined to read 10 TiB of data and write 1.35 TiB of data. A unique file was used by each process, and all data was perfectly aligned. The application was therefore able to sustain I/O for an extended period with no significant metadata or misalignment overhead. This job achieved the highest observed efficiency of jobs analyzed in the case studies.

I. Turbulence2

The Turbulence2 project illustrates another example where the job variability arose from differences in the performance of a single application at different scales. The jobs took very little run time, with several examples executing for less than one minute. There is an unusual mix of access sizes, as illustrated in Figure 9. Writes were dominated by very large access sizes, but many reads were less than 100 bytes each. This strategy performed best at relatively small job sizes of 2,048 or 2,304 processes. The same application did not fare was well when scaled up to 65,536 or 131,072 processes, though the run time was still only a few minutes. This application used MPI-IO but did not leverage collective operations or derived datatypes. The I/O-intensive jobs in this project may have been the result of a benchmarking effort, as only 40 jobs met the filtering criteria used in this section. All 40 were the same application executed with different parameters and at different scales.

J. Turbulence3

The Turbulence3 project consisted of 49 jobs. There were a few common sizes of 1,024, 1,600 and 2,048 processes as well as two 8,192 process jobs. The jobs have a similar pattern of I/O. There is a mix of MPI independent reads and writes at a specific request size. The I/O also occurs only from a subset of the MPI ranks, either 4 or 8 ranks. The lowest-performing job used 16 KiB MPI-IO independent reads and writes. The performance increased as jobs used larger request sizes, going up to 320 KiB and 512 KiB request sizes. The highest performing job used a 320 KiB request size but had more than double the number of reads as writes. The reads would be able to take advantage of GPFS read-ahead and caching.

VI. APPLICATION TUNING CASE STUDY

At the conclusion of the I/O study interval, we revisited the CombustionPhysics project, analyzed its behavior in greater detail with Darshan, and optimized it for improved I/O performance. As noted earlier, production jobs from this project ranged from 1,024 to 32,768 nodes but achieved progressively worse I/O performance as the scale increased. This application used OpenMP with four threads per node to process an AMR data set ranging from 2^{10} to 2^{13} data points. To simplify the tuning process, we decided to investigate the I/O of a similar but smaller application example from the CombustionPhysics project. This target application utilized the same OpenMP configuration and the same I/O stratey, and likewise achieved poor I/O performance at scale. However, its data set is a uniform 2^9 mesh that produces fixed-size checkpoints of approximately 20 GiB each. We focused on the 8,192 node (32,768 cores) example of this application as a test case and configured it to generate two checkpoints.

This application was originally tuned for a high-performance computing system that achieved optimal performance by using a unique file for the checkpoint data on each node. As in examples shown earlier in this work, however, we found that on Intrepid this approach led to I/O behavior that was dominated by metadata overhead. According to Darshan, the total I/O time to dump 40 GiB of data was approximately 728 seconds, nearly all of which was attributed to metadata activity. As an experiment, we precreated the output files for the application in order to observe the change in I/O behavior when file creation cost was eliminated from the run time. This approach reduced the total I/O time to 25 seconds, a factor of 28 improvement over the original execution. Such an approach is not feasible for production use, however: file precreation is

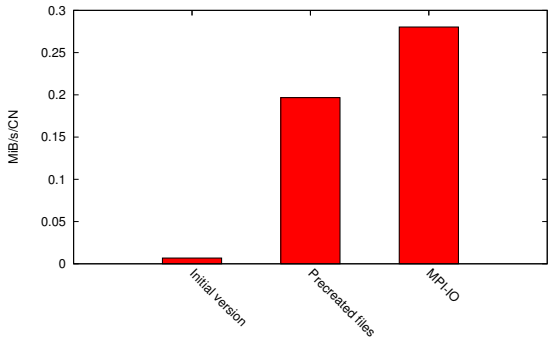

Fig. 11. CombustionPhysics per node performance with 8,192 nodes (32,768 cores).

a time-consuming process, and the number of checkpoints that will be generated is not known in advance.

To reduce the metadata overhead in a more practical manner, we decided to modify the application to dump its checkpoint data to a single, shared file. Rather than dumping data using independent POSIX operations, however, we updated the application to use MPI-IO collective operations. By doing so, the application not only reduced metadata overhead but also enabled a range of transparent MPI-IO optimizations. Of particular importance for strong scaling algorithms, MPI-IO can aggregate access to shared files in order to mitigate the affect of smaller writes at scale. Figure 11 shows the per node I/O performance achieved by all three versions of the application, using the same MiB/s/CN metric used in the two-month system study. The MPI-IO version achieved a factor of 41 improvement over the original application, dumping two time steps in approximately 17 seconds. Darshan analysis confirmed that all application-level writes were between 1 and 4 MiB in size, as in the original example. However, MPI-IO aggregated these into 16 MiB writes at the file system level. The total number of write operations processed by the file system was therefore reduced from 16,388 to 4,096, resulting in more efficient use of available resources and faster turnaround for science runs.

VII. Related work

A number of past studies have investigated the I/O access patterns of scientific applications. Nieuwejaar et al. initiated the influential Charisma project in 1993 to study multiprocessor I/O workloads [13]. This culminated in an analysis of three weeks of data from two high-performance computing systems with up to 512 processes. Their study identified access pattern characteristics and established terminology to describe them. Smirni and Reed analyzed five representative scientific applications with up to 64 processes [14]. The Pablo environment [15] was used for trace capture in that work. While both Charisma and Pablo measured application characteristics in a manner similar to our work, neither was performed at a comparable scale or correlated to system-level I/O activity. Wang et al. investigated synthetic benchmarks and two physics applications with up to 1,620 processes and found similar results [16]. Uselton et al. developed a statistical approach to I/O characterization in a more recent study [17]. They leveraged IPM [18] for the raw trace capture of two scientific applications with up to 10,240 processes on two different platforms. Statistical techniques were then used to identify and resolve I/O bottlenecks in each application. These studies utilized complete traces that focused on specific applications rather than on a general production workload.

Other recent system-level studies have focused on large network file systems. In an investigation of two CIFS file systems that hosted data for 1,500 industry employees, Leung et al. discovered a number of recent trends in I/O behavior [6]. Anderson presented a study of NFS workloads with up to 1,634 clients [5]. These studies were similar in scope to our work but were not performed in a high-performance computing environment.

A wide variety of tools are available for capturing and analyzing I/O access from individual parallel applications, including IPM, HPCT-IO, LANL-Trace, IOT, and mpiP [18], [19], [20], [21], [22]. Multiple I/O tracing mechanisms were surveyed by Konwinski et al [23]. Klundt, Weston, and Ward have also investigated tracing of user-level I/O libraries on lightweight kernels [24].

VIII. Conclusions

In this work we have investigated critical questions about the nature of storage access characteristics on leadership-class machines. We performed our investigation over two months of production time on Intrepid, a 557-teraflop IBM Blue Gene/P deployed at Argonne National Laboratory. Intrepid's storage system contained over 191 million files and moved an average of nearly 150 TiB of data per day. We captured detailed application-level I/O characteristics of 27% of all jobs executed on Intrepid, ranging in size from 1 to 163,840 processes. In doing so, we demonstrated that it is possible to instrument several aspects of storage systems at full scale without interfering with production users. We have also developed a performance metric that enables relative comparison of a wide variety of production applications.

The results of this study have led to findings that will influence future research direction as well as design of future I/O subsystems to be used at the ALCF. We found that POSIX is still heavily used by many applications, though on this system there was no discernable performance advantage in that choice. MPI-IO, HDF5, and Parallel NetCDF are also used by the top 10 I/O producers and consumers. The ALCF will need to continue providing support for all these interfaces.

Shared or partially shared file usage becomes the predominant method of file access at the 16,384-processor mark on Intrepid. This job size, and larger, will be increasingly common on the next generation of computing platforms. This implies that the ALCF should invest in helping applications transition to shared or paritially shared file models. We demonstrated the benefit of this approach through a case study of I/O tuning in the CombustionPhysics project, one of the most active INCITE projects on Intrepid in terms of data usage.

From the aspect of I/O system design, we found two major items of interest. We were able to verify the "burstiness" of I/O on Intrepid, indicating that we have a significant opportunity to utilize idle storage resources for tasks such as performance diagnosis. In addition, we found that files are rarely overwritten once they are closed. This suggests that there is an opportunity to leverage hierarchical storage for more cost-effective storage of infrequently accessed data. We also found that while I/O characterization data was easy to use on a per job basis, analyzing and summarizing many jobs in aggregate were more difficult than anticipated. As a result, we have enhanced Darshan in the 2.0.0 release to streamline the analysis process [2]. For example, more shared file statistics (such as minimum, maximum, and variance among participating processes) are now computed at run time.

We compared storage access characteristics of different scientific application domains and found an extraordinary variety of both data usage and application-level I/O performance. Several distinct I/O strategies were identified, including shared file usage, unique file usage, rank 0 I/O, and examples of both weak and strong scaling of data. We also discovered examples of applications that appeared to be utilizing custom aggregation algorithms without the assistance of MPI-IO. In general we found that metadata overhead, small access sizes, and I/O imbalance were the most significant barriers to I/O performance. However, no single technique employed by applications emerged overwhelmingly as the most successful. In future work we would like to perform similar studies at other sites that may offer a different collection of scientific computing applications to compare and contrast.

ACKNOWLEDGMENTS

This work was supported by Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract DE-AC02-06CH11357.

This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

REFERENCES

[1] S. Lang, P. Carns, R. Latham, R. Ross, K. Harms, and W. Allcock, "I/O performance challenges at leadership scale," in *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. New York, NY, USA: ACM, 2009, pp. 1–12.

[2] G. R. Ganger, "Generating representative synthetic workloads: An unsolved problem," in *Proceedings of the Computer Measurement Group (CMG) Conference*, 1995, pp. 1263–1269.

[3] A. Traeger, E. Zadok, N. Joukov, and C. P. Wright, "A nine year study of file system and storage benchmarking," *Trans. Storage*, vol. 4, no. 2, pp. 1–56, 2008.

[4] N. Agrawal, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Towards realistic file-system benchmarks with CodeMRI," *SIGMETRICS Perform. Eval. Rev.*, vol. 36, no. 2, pp. 52–57, 2008.

[5] E. Anderson, "Capture, conversion, and analysis of an intense NFS workload," in *FAST '09: Proccedings of the 7th conference on File and storage technologies*. Berkeley, CA, USA: USENIX Association, 2009, pp. 139–152.

[6] A. W. Leung, S. Pasupathy, G. Goodson, and E. L. Miller, "Measurement and analysis of large-scale network file system workloads," in *Proceedings of the 2008 USENIX Technical Conference*. Berkeley, CA, USA: USENIX Association, 2008, pp. 213–226. [Online]. Available: http://portal.acm.org/citation.cfm?id=1404014.1404030

[7] P. Carns, R. Latham, R. Ross, K. Iskra, S. Lang, and K. Riley, "24/7 characterization of petascale I/O workloads," in *Proceedings of 2009 Workshop on Interfaces and Architectures for Scientific Data Storage*, September 2009.

[8] "U.S. Department of Energy INCITE program," http://www.er.doe.gov/ascr/incite/.

[9] F. Schmuck and R. Haskin, "GPFS: A shared-disk file system for large computing clusters," in *Proceedings of the FAST 2002 Conference on File and Storage Technologies*, January 2002.

[10] S. Godard, "SYSSTAT utilities home page," http://pagesperso-orange.fr/sebastien.godard/.

[11] S. Dayal, "Characterizing HEC storage systems at rest," Carnegie Mellon University Parallel Data Lab, Tech. Rep. CMU-PDL-08-109, 2008.

[12] W. Liao and A. Choudhary, "Dynamically adapting file domain partitioning methods for collective I/O based on underlying parallel file system locking protocols," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. IEEE Press, Piscataway, NJ, 2008.

[13] N. Nieuwejaar, D. Kotz, A. Purakayastha, C. S. Ellis, and M. Best, "File-access characteristics of parallel scientific workloads," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 10, pp. 1075–1089, October 1996. [Online]. Available: http://www.computer.org/tpds/td1996/l1075abs.htm.

[14] E. Smirni and D. Reed, "Workload characterization of input/output intensive parallel applications," in *Proceedings of the Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, ser. Lecture Notes in Computer Science, vol. 1245. Springer-Verlag, June 1997, pp. 169–180. [Online]. Available: http://vibes.cs.uiuc.edu/Publications/Papers/Tools97.ps.gz

[15] D. A. Reed, R. A. Aydt, R. J. Noe, P. C. Roth, K. A. Shields, B. W. Schwartz, and L. F. Tavera, "Scalable performance analysis: The Pablo performance analysis environment," in *Proceedings of the Scalable parallel libraries conference*. IEEE Computer Society, 1993, pp. 104–113.

[16] F. Wang, Q. Xin, B. Hong, S. A. Brandt, E. L. Miller, D. D. E. Long, and T. T. Mclarty, "File system workload analysis for large scale scientific computing applications," in *Proceedings of the 21st IEEE / 12th NASA Goddard Conference on Mass Storage Systems and Technologies*, 2004, pp. 139–152.

[17] A. Uselton, M. Hawison, N. Wright, D. Skinner, J. Shalf, L. Oliker, N. Keen, and K. Karavanic, "Parallel I/O performance: From events to ensembles," in *Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium (To Appear)*, 2010.

[18] N. J. Wright, W. Pfeiffer, and A. Snavely, "Characterizing parallel scaling of scientific applications using IPM," in *The 10th LCI International Conference on High-Performance Clustered Computing*, 2009.

[19] S. Seelam, I.-H. Chung, D.-Y. Hong, H.-F. Wen, and H. Yu, "Early experiences in application level I/O tracing on Blue Gene systems," in *Proceedings of the 2008 IEEE International Parallel and Distributed Processing Symposium*, 2008.

[20] "HPC-5 open source software projects: LANL-Trace," http://institute.lanl.gov/data/software/#lanl-trace.

[21] P. C. Roth, "Characterizing the I/O behavior of scientific applications on the Cray XT," in *PDSW '07: Proceedings of the 2nd International Workshop on Petascale Data Storage*. New York, NY, USA: ACM, 2007, pp. 50–55.

[22] J. S. Vetter and M. O. McCracken, "Statistical scalability analysis of communication operations in distributed applications," *SIGPLAN Notices*, vol. 36, no. 7, pp. 123–132, 2001.

[23] A. Konwinski, J. Bent, J. Nunez, and M. Quist, "Towards an I/O tracing framework taxonomy," in *PDSW '07: Proceedings of the 2nd international workshop on Petascale data storage*. New York, NY, USA: ACM, 2007, pp. 56–62.

[24] R. Klundt, M. Weston, and L. Ward, "I/O tracing on Catamount," Sandia National Laboratory, Tech. Rep. SAND2008-3684, 2008.

[2] http://www.mcs.anl.gov/research/projects/darshan/